Hochiminh City University of Technology
Computer Science and Engineering
[CO1027] - Fundamentals of C++ Programming

Array, Pointer, Structure

Lecturer: Duc Dung Nguyen
Credits: 3

Outcomes

- Using array, and structure data types
- Using pointer
- Allocating and releasing dynamic memory

Today's outline

- Structured data types
  - Array
  - Structure
- Pointer
- Dynamic memory

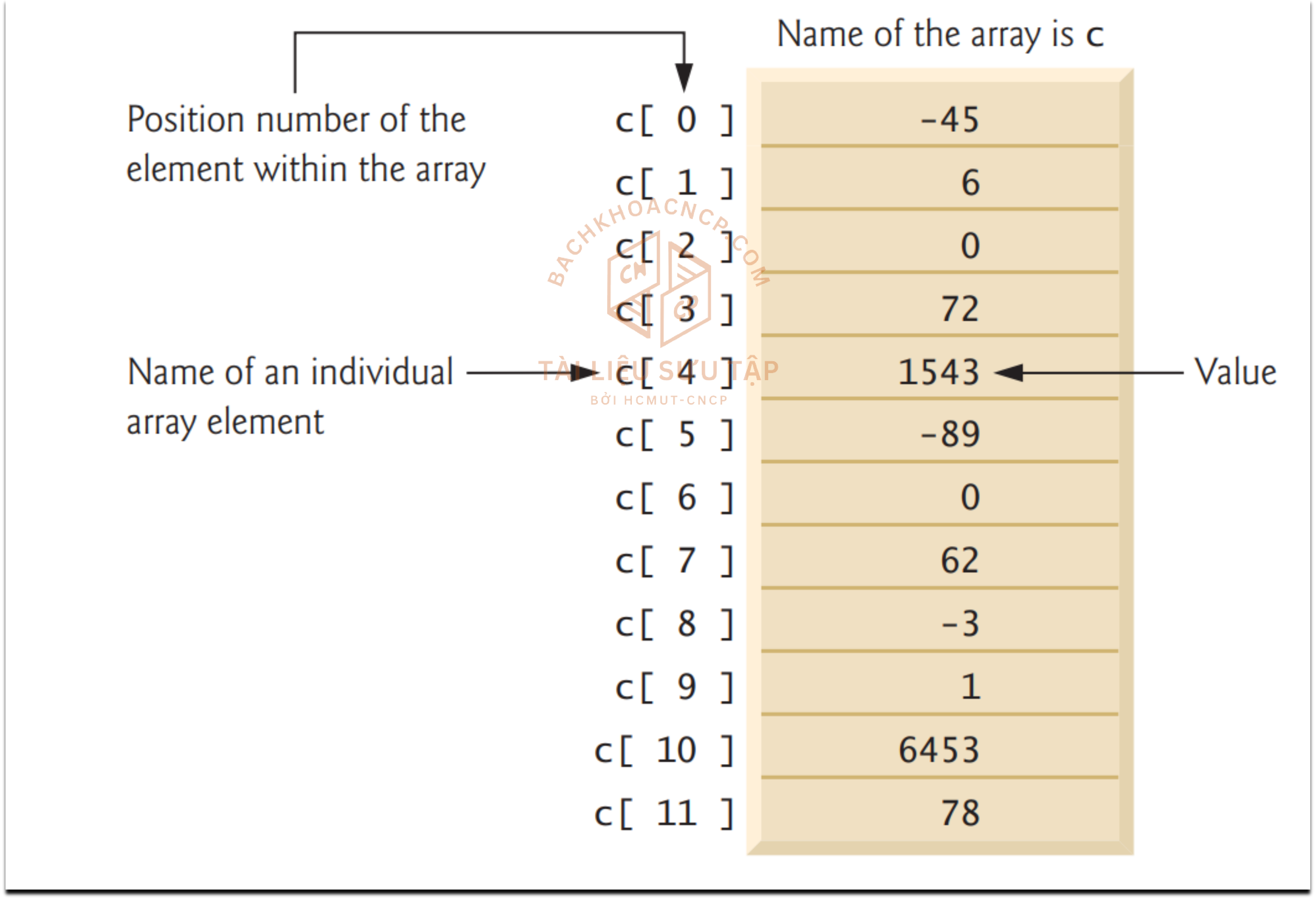
Structured data types

Structured data types

- Can we implement a program with only basic data types?
- What do we need beside basic data types?
  - A sequence of memory slots that contains a specific data type
  - A mixture of different data types

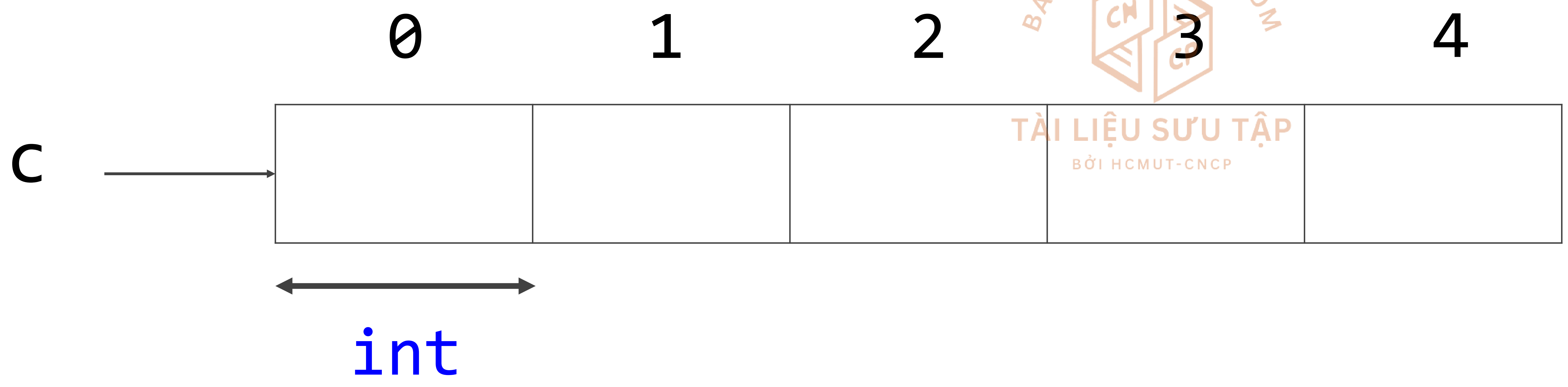
Array

Array


Name of the array is c
Position number of the element within the array
Name of an individual array element
Value
c[0]
-45
c[1]
6
c[2]
0
c[3]
72
c[4]
1543
c[5]
-89
c[6]
0
c[7]
62
c[8]
-3
c[9]
1
c[10]
6453
c[11]
78

Declaring Arrays

- <type> *arrayName* [*arraySize*];
  - E.g: `int c[5];`



Initializing Arrays

- One by one
  - E.g: `c[4] = 10;`
- Using a loop
  - E.g: `for (int i = 0; i < 5; i++) c[i] = 0;`
- Declaring with an initializer list
  - E.g: `int c[5] = { 10, 20, 30, 40, 50 };`

Accessing the values of an array

- The values of any of the elements in an array can be accessed just like the value of a regular variable of the same type. The syntax is:
  - name[index]
- Example:

```
int a = 2;
c[0] = a;
c[a] = 75;
b = c[a + 2];
c[c[a]] = c[2] + 5;
```

Example

```
#include<iostream>
#include<iomanip>
using namespace std;

int main() {
    int c[10];
    for (int i = 0; i < 10; i++) c[i] = 0;

    cout << "Element" << setw(13) << "Value" << endl;
    for (int i = 0; i < 10; i++)
        cout << setw(7) << i << setw(13) << c[i] << endl;

    return 0;
}
```

Example

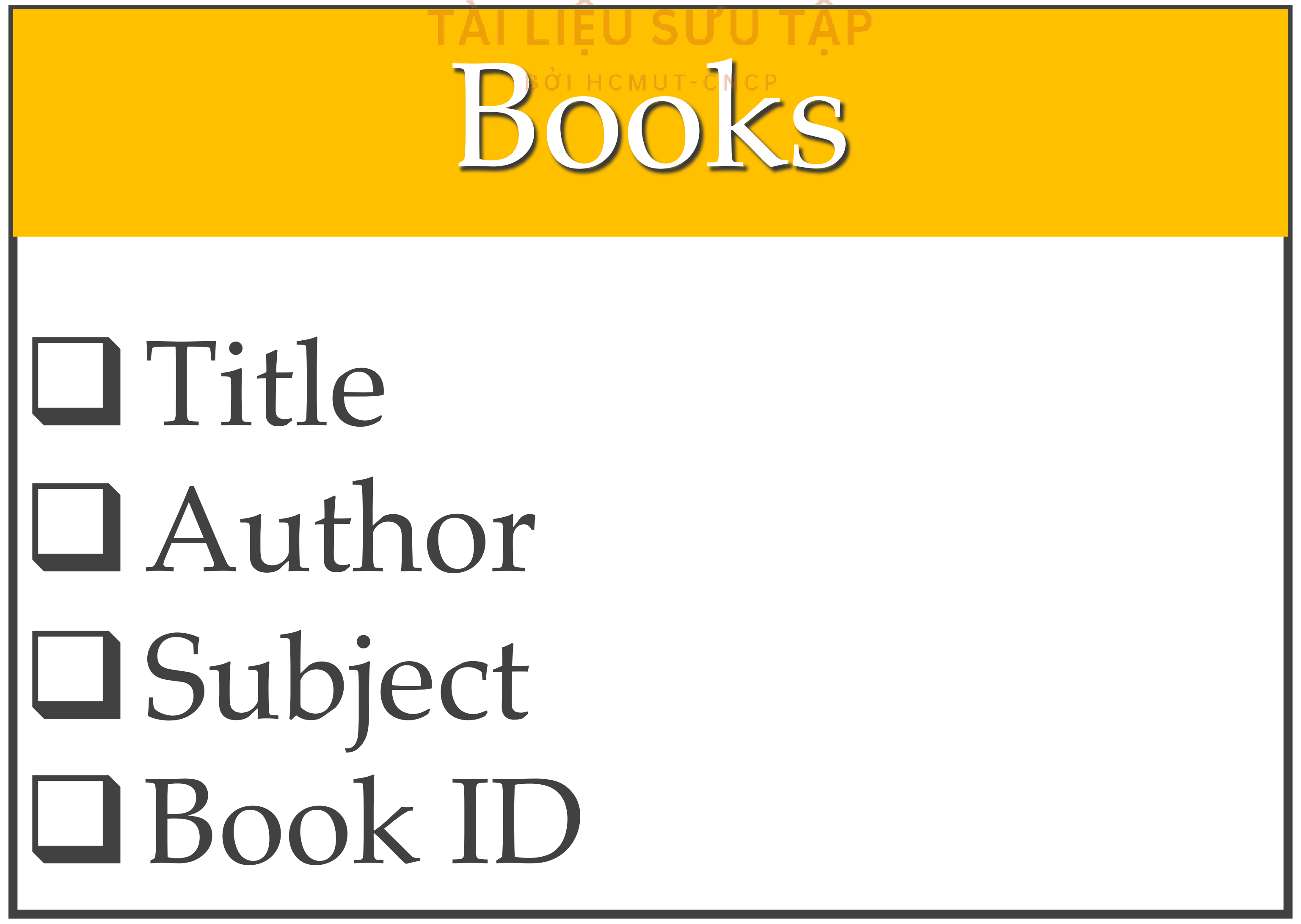
```
#include<iostream>
#include<iomanip>
using namespace std;

int main() {
   int c[10] = { 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 };

   cout << "Element" << setw(13) << "Value" << endl;
   for (int i = 0; i < 10; i++)
      cout << setw(7) << i << setw(13) << c[i] << endl;

   return 0;
}
```

Structure

Structure

- Structure is user defined data type which allows you to combine data items of different kinds.

- Structures are used to represent a record.



Books

- □ Title
- □ Author
- □ Subject
- □ Book ID

Defining a Structure

```
struct [structure tag] {
   member definition;
   member definition;
   ...
   member definition;
}[structure variable(s)];
```

Example

```
struct Books {
  char  title[50];
  char  author[50];
  char  subject[100];
  int   book_id;
};

struct Books book1;
```

The typedef Keyword

- “Aliasing” types defined by user using keyword typedef

```
typedef struct {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
} Books;

Books book1;
```

Accessing Structure Members

- Using the member access operator (.)

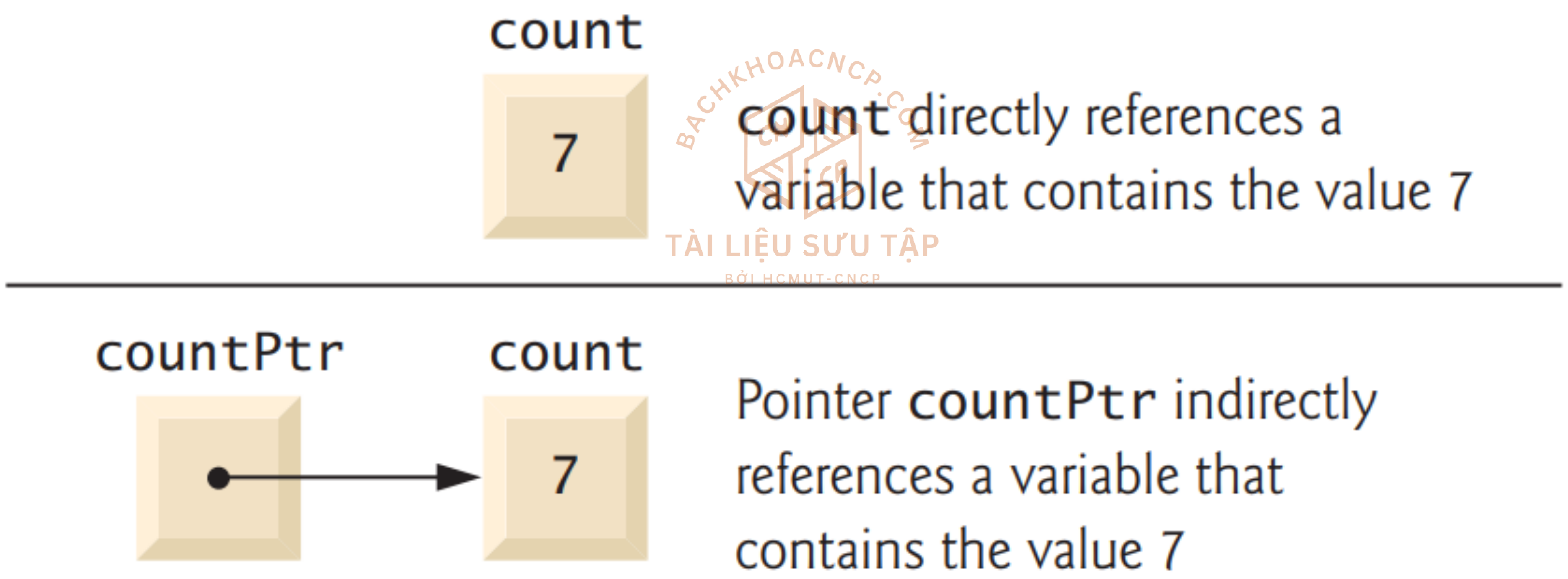
```
struct Books Book1;        // Declare Book1 of type Book

// book 1 specification
strcpy_s(Book1.title, "Learn C++ Programming");
strcpy_s(Book1.author, "Chand Miyan");
strcpy_s(Book1.subject, "C++ Programming");
Book1.book_id = 6495407;
```

https://www.tutorialspoint.com/cplusplus/cpp_data_structures.htm

Pointer

Pointer

- Pointer variables contain memory addresses as their values



count directly references a variable that contains the value 7

Pointer countPtr indirectly references a variable that contains the value 7

Declaring Pointers

- Direct way:
  - <type> * <identifier>;
  - E.g.: `int * a;`
- Using typedef keyword:
  - typedef <type>* <alias_type>;
  - E.g.:

```
typedef int * intPointer;
intPointer a;
```

Using Pointers

- Defining a pointer variable

```
int  var = 20;   // actual variable declaration.
int  *ip;        // pointer variable
```

- Assigning the address of a variable to a pointer

```
ip = &var;       // using address operator &
```

- Accessing the value at the address available in the pointer variable

```
cout << *ip << endl; // using dereferencing operator *
*ip = 5;
```

NULL Pointers

- Assigning the pointer NULL to a pointer variable in case you do not have exact address to be assigned.

```
int  *ptr = NULL;
```

- To check for a null pointer you can use an if statement as follows

```
if(ptr == NULL)

Or

if(!ptr)
```

Using const with Pointers

- ***Nonconstant Pointer to Nonconstant Data***
- ***Nonconstant Pointer to Constant Data***
- ***Constant Pointer to Nonconstant Data***
- ***Constant Pointer to Constant Data***

Using const with Pointers

- ***Nonconstant Pointer to Nonconstant Data***
  - the data can be modified through the dereferenced pointer
  - the pointer can be modified to point to other data

```
int a = 5;
int b = 9;
int *ptr = &a;
*ptr = 6; // OK
ptr = &b; // OK
```

Using const with Pointers

- ***Nonconstant Pointer to Constant Data***
  - the data can NOT be modified through the dereferenced pointer
  - the pointer can be modified to point to other data

```
const int a = 5;
int b = 9;
const int *ptr = &a;
*ptr = 6; // Error
ptr = &b; // OK
```

Using const with Pointers

- ***Constant Pointer to Nonconstant Data***
  - the data can be modified through the dereferenced pointer
  - the pointer can NOT be modified to point to other data

```
int a = 5;
int b = 9;
int* const ptr = &a;
*ptr = 6; // OK
ptr = &b; // Error
```

Using const with Pointers

- ***Constant Pointer to Constant Data***
  - the data can NOT be modified through the dereferenced pointer
  - the pointer can NOT be modified to point to other data

```
const int a = 5;
const int b = 9;
int* const ptr = &a;
*ptr = 6; // Error
ptr = &b; // Error
```

Pointer Arithmetic

- Four arithmetic operators that can be used on pointers: ++, --, +, and -

```
// point to the next location
ptr++;
ptr = ptr + 1;

// point to the previous location
ptr--;
ptr = ptr - 1;
```

Pointer

- Order of operators

```
*p++    // *(p++)
*++p    // *(++p)
++*p    // ++(*p)
(*p)++ // increase value at location pointed by p
       // (only valid with integer pointers)
```

When in doubt, use safe statement.

Pointers vs Arrays

- Arrays and pointers are intimately related in C++ and may be used *almost* interchangeably.

```
int  var[MAX] = { 10, 100, 200 };
int  *ptr;

// let us have array address in pointer.
ptr = var;
```

- However, an array name can be thought of as a constant pointer.

```
var++; // This is incorrect syntax.
```

Example

```
#include <iostream>
using namespace std;
const int MAX = 3;

int main() {
    int  var[MAX] = { 10, 100, 200 };
    int  *ptr;

    // let us have array address in pointer.
    ptr = var;

    for (int i = 0; i < MAX; i++) {
        cout << "Address of var[" << i << "] = " << ptr << endl;
        cout << "Value of var[" << i << "] = " << *ptr << endl;

        // point to the next location
        ptr++;
    }
    return 0;
}
```

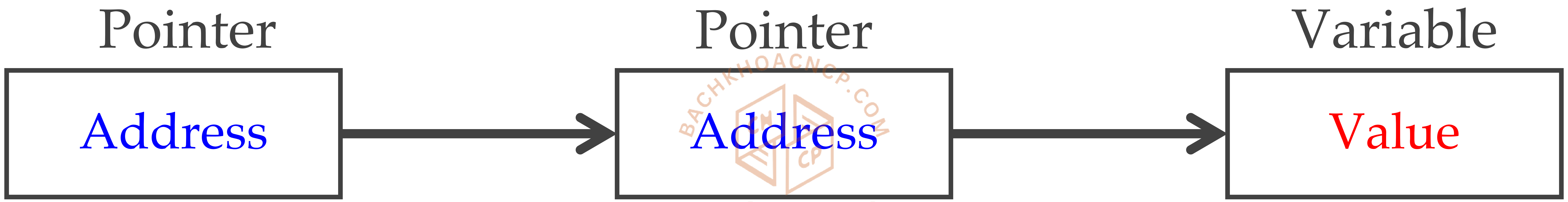
https://www.tutorialspoint.com/cplusplus/cpp_pointer_arithmatic.htm

Array of Pointers

```
#include <iostream>

using namespace std;
const int MAX = 4;

int main() {
   const char *names[MAX] = { "An Nguyen", "Binh Tran", "Cong Pham", "Dat Le" };

   for (int i = 0; i < MAX; i++) {
      cout << "Value of names[" << i << "] = " << *(names + i) << endl;
   }

   return 0;
}
```

https://www.tutorialspoint.com/cplusplus/cpp_array_of_pointers.htm

Pointer to Pointer



```
int  var = 300;
int  *ptr = &var; // take the address of var
int  **pptr = &ptr; // take the address of ptr
```

https://www.tutorialspoint.com/cplusplus/cpp_pointer_to_pointer.htm

Example

```
#include <iostream>

using namespace std;

int main() {
    int  var = 300;
    int  *ptr = &var; // take the address of var
    int  **pptr = &ptr; // take the address of ptr

    // take the value using pptr
    cout << "Value of var :" << var << endl;
    cout << "Value available at *ptr :" << *ptr << endl;
    cout << "Value available at **pptr :" << **pptr << endl;

    return 0;
}
```

https://www.tutorialspoint.com/cplusplus/cpp_pointer_to_pointer.htm

References

- A reference variable is an alias, that is, another name for an already existing variable.

```
int a = 10;
int& b = a;
```

- References are usually used for function argument lists and function return values.

https://www.tutorialspoint.com/cplusplus/cpp_references.htm

Example

```
#include<iostream>
using namespace std;

int main() {
   int a = 10;
   int& b = a;
   cout << "Value of a :" << a << endl;
   cout << "Value of a reference :" << b << endl;
   a = 6;
   cout << "Value of a :" << a << endl;
   cout << "Value of a reference :" << b << endl;
}
```

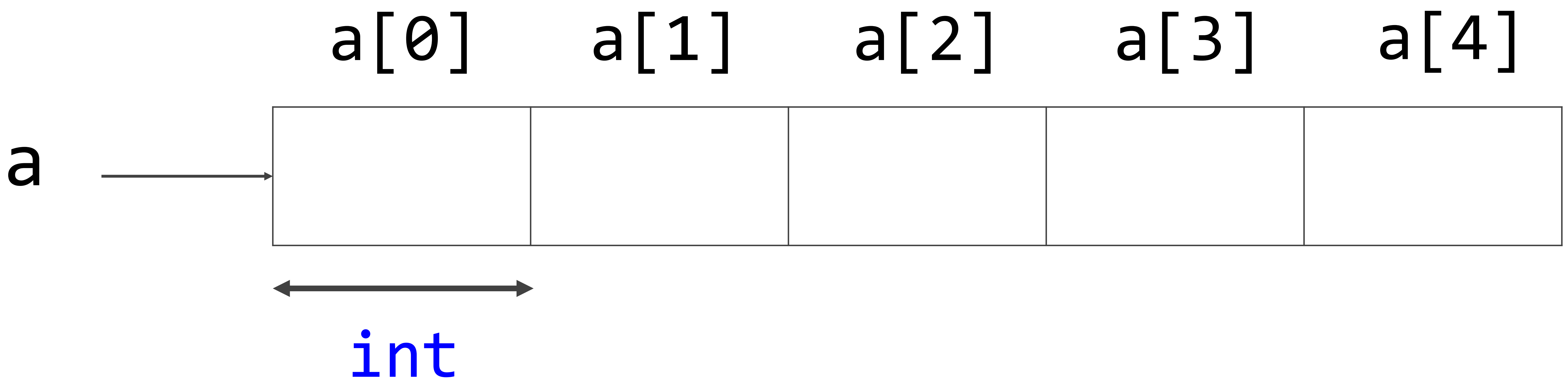
Pointers vs References

- Three major differences between references and pointers:
  - You cannot have NULL references. You must always be able to assume that a reference is connected to a legitimate piece of storage.
  - Once a reference is initialized to an object, it cannot be changed to refer to another object.
  - A reference must be initialized when it is created. Pointers can be initialized at any time. Pointers can be pointed to another object at any time.

Dynamic Merory

Allocating Dynamic Memory

- Dynamic memory is allocated using operator `new.` Here is the syntax:
  - <pointer> = `new` <type>
  - <pointer> = `new` <type> [<number_of_elements>]
- Examples:
  - `int * a = new int[5];`



Releasing Dynamic Memory

- Dynamic memory is released using operator `delete.` Here is the syntax:
  - `delete` <pointer>
  - `delete []` <pointer>
- Examples:
  - `delete []a;`

Summarise

- Structured data types
  - Array
  - Structure
- Pointer
- Dynamic memory